Računalni inteligentni igrač za igranje igre Poker

Širić, Fran

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1694

RAČUNALNI INTELIGENTNI IGRAČ ZA IGRANJE IGRE POKER

Fran Širić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1694

RAČUNALNI INTELIGENTNI IGRAČ ZA IGRANJE IGRE POKER

Fran Širić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1694

Pristupnik:	**Fran Širić (0036540591)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	izv. prof. dr. sc. Marko Čupić

Zadatak: **Računalni inteligentni igrač za igranje igre Poker**

Opis zadatka:

Kartaška igra Poker pripada u igre s nepotpunom informacijom. Postoji nekoliko inačica ove igre i razvoj kvalitetnog računalnog igrača nije jednostavan zadatak. U okviru ovog završnog rada potrebno istražiti pristupe izgradnje računalnog inteligentnog igrača igre poker, odabrati jednu inačicu te napraviti prototipnu programsku implementaciju. Opisati i vrednovati razvijeno programsko rješenje. Radu priložiti izvorni i izvršni kod, uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

Uvod

Umjetna inteligencija i neuronske mreže su tehnologija koja se zadnjih godina sve brže razvija i raste u upotrebi. Koristi se u mnogo industrija i ima široke namjene. Jedna od namjena je i uloga inteligentnog agenta u video igrama. Najčešća pojava umjetne inteligencije je u ulozi pametnog protivnika ili NPC-a (lika kojim ne upravlja igrač). Takve umjetne inteligencije su najčešće dizajnirane koristeći produkcijska pravila, kondicionalne tvrdnje ili algoritme pretraživanja. To se postiže korištenjem poznatog znanja o svijetu, okruženju i mogućim stanjima u kojima se agent može naći. Stanja nekad može biti vrlo mnogo te postane vrlo teško odrediti sva stanja i kako njima rukovati.

Umjetna inteligencija za igranje igrice se također može postići i korištenjem neuronskih mreža. Neuronske mreže imaju mogućnost preći preko nekih ograničenja klasične umjetne inteligencije u video igrama. Problem stanja postaje mnogo jednostavniji zbog sposobnosti neuronske mreže da se poboljšava i sama prilagođava stanju u kojem se nalazi. Postoje i problemi. Ovisno o igri, nekada je teško stvoriti dobro okruženje za treniranje, a nekad se mreža ne ponaša kako je predviđeno.

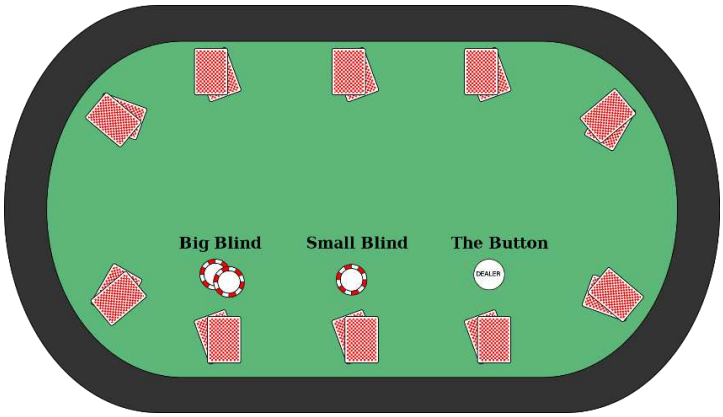
Napravljena je neuronska mrežu koja igra video igru pokera u Texas Hold'em varijanti, korištenjem genetskog algoritma. U ovom radu je analiziran njen dizajn, treniranje i funkcionalnost te prednosti i nedostatci mreže trenirane genetskim algoritmom na igru pokera.

1. Igra Texas Hold'em poker

Implementirana je poker igra sa Texas Hold'em pravilima. Na toj implementaciji igre je i trenirana mreža. Ljudski igrač također može igrati igricu protiv neuronske mreže ili protiv demo igrača.

1.1. Pravila

Texas Hold'em poker je varijanta pokera gdje svaki igrač u ruku dobiva dvije karte koje drugi igrači ne vide. Igrač koji počinje je prvi po redu nakon igrača sa oznakom dealera u smjeru kazaljke na satu (small blind). Mora uložiti predodređeni ulog, a igrač koji je prvi po redu nakon njega (big blind) mora uložiti dvostruko veći ulog. Uloge i pozicija za stolom se može vidjeti na slici (Sl. 1.1 - Stol i raspored uloga).



Sl. 1.1 - Stol i raspored uloga [1]

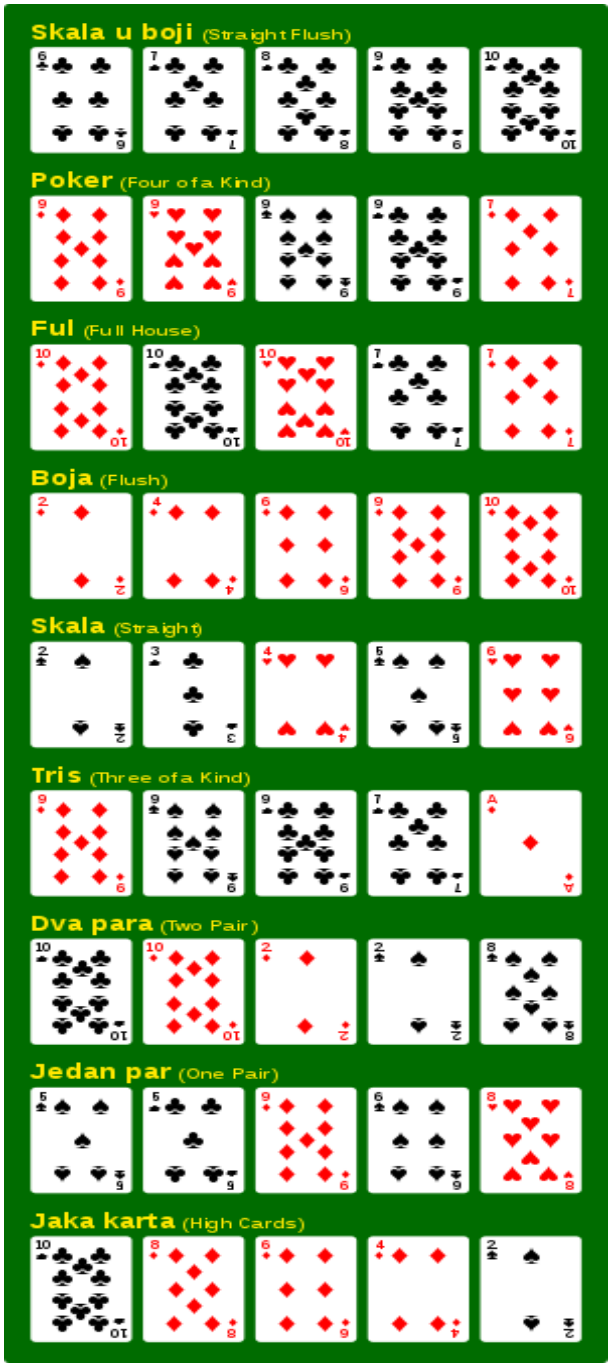
Svaki igrač nadalje ima priliku ili odustati (fold), ne uložiti ništa ako nitko nije povisio ulog (check), uložiti jednako najvećem ulogu (call), ili povisiti ulog (raise). Ide se u krug i svaki igrač mora napraviti potez, dok svi igrači nisu napravili potez i svi igrači koji su u igri nisu uložili jednako najvišem ulogu ili uložili sve čipove koje imaju. Tada se izvlače tri karte na stol koje svi igrači vide. Ponavlja se isti proces poteza, te se izvlači još jedna karta. Naposljetku se opet ponavlja proces poteza, izvlači se zadnja karta i igrači na kraju imaju priliku za zadnji potez.

Kada su svi igrači napravili zadnji potez tada otkrivaju svoje karte, i igrač koji ima najsnažniju kombinaciju 5 karata, od 7 karata iz njegove ruke i sa stola, pobjeđuje.

Pobjednik uzima ulog od svakog igrača jednak svom ulogu. Igrači koji slijede po jačini nakon pobjednika rade isto dok sav ulog na stolu nije podijeljen. Sljedeći igrač po redoslijedu dobiva dealer oznaku i igra se nastavlja.

1.1.1. Jačine ruku

Od sedam karata (igračeve dvije i pet na stolu) uvijek se gleda kombinacija od 5 karata sa najvećom vrijednosti. Vrijednost se određuje sa definiranim kombinacijama [2] koje se mogu vidjeti na slici (Sl. 1.2 - Dobitne kombinacije u pokeru).



Sl. 1.2 - Dobitne kombinacije u pokeru [2]

Najsnažnija kombinacija je poredak pet uzastopnih karata u istoj boji, skala u boji. Slijedi četiri iste karte, poker. Nakon toga ful, kombinacija od tri iste i još drugačije dvije iste karte, ful. Nakon fula je najsnažnija kombinacija od pet karata iste boje, boja. Nakon boje je najsnažniji poredak od pet uzastopnih karata skala. Zatim tri iste karte, tris. Na kraju su dva para, jedan par i ako nema kombinacije, gleda se najjača karta.

U slučaju da postoje dva ili više igrača sa jednakom snagom ruke, svakom igraču se dijeli jednaki udio dobitka. U slučaju da dva igrača imaju istu kombinaciju za koju ne treba svih pet karata, naprimjer dva para, tada jaču ruku ima igrač koji ima veću preostalu kartu.

1.1.2. Cilj

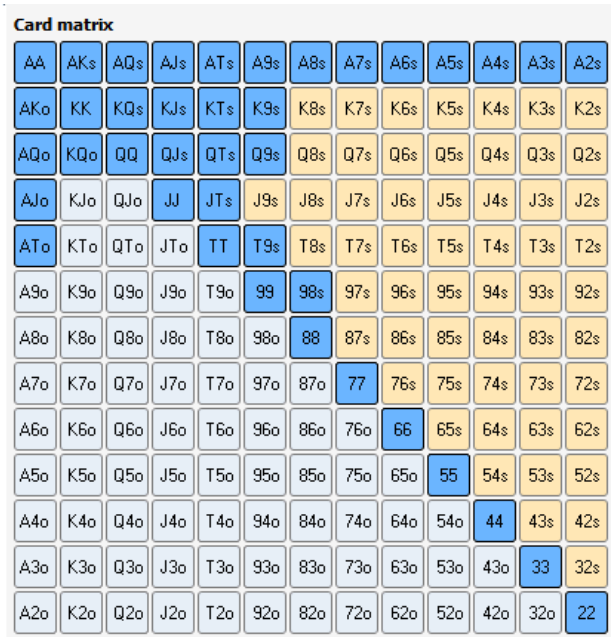
Poker završava kada ostane jedan igrač a svi ostali nemaju čipova za uložiti, ili po dogovoru igrača. Igrač koji je završio sa najviše čipova je pobjednik. Poker se može igrati na način da igrači mogu doći i sjesti za stol kada žele, i razmijeniti čipove koje su osvojili za novce, ili u formatu turnira igrač ostaje u igri dok god ima čipova, i kada je izbačen jer je izgubio sve čipove dobiva nagradu ovisno o mjestu. U implementaciji igre koju će igrati neuronska mreža, igra će se igrati dok ima više od jednog igrača sa više od 0 čipova ili dok se ne odigra 50 rundi.

1.2. Strategija

Poker je igra sa mnogo stanja i varijabli. Trebaju se uzeti u obzir karte u ruci, karte na ploči, pozicija za stolom, količina čipova i još mnogo faktora. Iako se na prvu možda čini da je poker igra gdje prevladava sreća, teorija pokera je vrlo duboka i kompleksna, te će iskusan i vješt igrač uvijek imati prednost.

Fundamentalni teorem pokera prema Davidu Sklansky-u kaže: *„Svaki put kada odigraš potez drugačije nego što bi ga odigrao da vidiš sve karte protivnika, oni dobivaju; i svaki put kada odigraš potez jednako kao što bi ga odigrao da vidiš sve njihove karte, oni gube“*[3]. Ono što je Sklansky ovim htio reći je vrlo jednostavno. Ako igrač vidi da ima najsnažniju ruku, tada će podići ulog, a ako vidi da nema najsnažniju ruku tada će odustati. To je pobjednička strategija igranja pokera.

Nažalost, u igri pokera se ne mogu vidjeti karte drugih igrača. Zato postoje mnoge statističke analize idealnih poteza u određenim situacijama kao na slici(Sl. 1.3 - Idealni potez prije izvlačenja karata). Slika prikazuje tablicu sa idealnim potezom za svaku kombinaciju karata u ruci, prije izvlačenja karata na stol. Plava polja označavaju dizanje uloga, svijetlo plava držanje pri nižem ulogu te narančasta moguće odustajanje. Treba uzeti u obzir da je ova tablica namijenjena samo jednoj poziciji na stolu, i postoje tablice za svaku poziciju i svaku rundu otkrivanja karata.



Sl. 1.3 - Idealni potez prije izvlačenja karata [4]

1.2.1. Optimalna teorija igre pokera

Pretpostavimo da postoji statistički savršen igrač koji uvijek donosi statistički savršenu odluku. Matematičar John Nash je dokazao da za svaku igru sa N igrača postoji strategija za koju vrijedi da se ne može dobiti bolji rezultat igre mijenjajući strategiju, pod uvjetom da ostali igrači ne mijenjaju strategiju[5]. U pokeru se to zove optimalna teorija igre (GTO)[4] pokera. U igri gdje sudjeluju takav statistički savršen igrač, i drugi igrači koji stalno koriste istu strategiju, statistički savršen igrač bi dobio ili ostao na nuli.

U pravoj igri sa ljudskim igračima optimalna strategija pokera nije idealna. Iako optimalna strategija igra „optimalno“, to znači da igrana na optimalan način u svrhu minimiziranja gubitka. Ljudski igrači međutim ne pokušavaju minimizirati gubitak, nego iskoristiti greške protivnika. Nadalje, ljudski igrači ne koriste istu strategiju cijelo vrijeme nego se prilagođavaju protivnicima. U tom slučaju statistički optimalna strategija sigurno ne bi nikada izašla kao gubitnik, ali vjerojatno ne bi izašla ni kao pobjednik.

1.2.2. Umjetna inteligencija kao poker igrač

Umjetna inteligencija koja igra poker i pokušava iskoristiti greške protivnika već postoji. Pluribus[6] je prva poker umjetna inteligencija koja je pobijedila ljude u kompleksnim online natjecanjima.

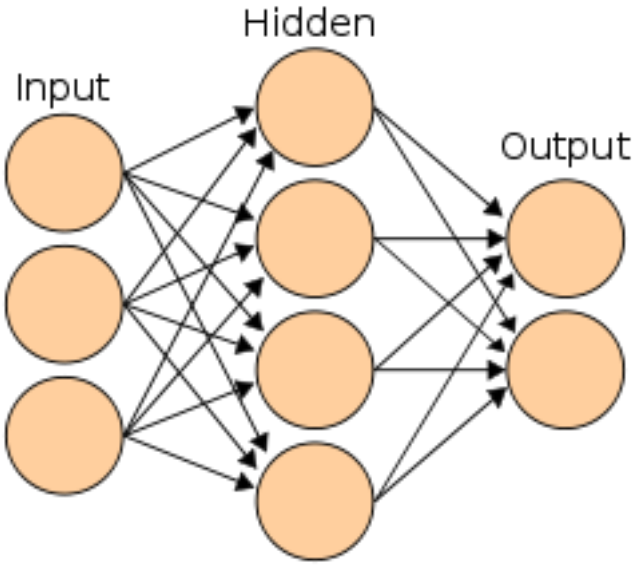
Offline je naučila osnovne strategije, te je zatim online u igrama protiv ljudskih protivnika nastavila učiti. Kako bi mreža naučila iskorištavati greške ljudskih protivnika, nužno je učiti ju dok igra protiv ljudskih protivnika. Umjetna inteligencija koja uči samo na statističkim podatcima može naučiti samo strategiju protiv gubitka, a ne strategiju dobitka.

Neuronska mreža koja je implementirana kroz ovaj rad neće učiti protiv ljudskih igrača, nego protiv drugih neuronskih mreža. Mreža nije učena na statističkim podacima i nema nikakvo znanje o igri. U ovoj situaciji najbolja strategija je optimalna teorija, iz razloga što svaka neuronska mreža ima konstantnu strategiju.

2. Neuronska mreža

2.1. Struktura

Neuronska mreža je sustav povezanih neurona. Ti povezani neuroni kroz međusobnu interakciju mogu činiti niz operacija. Neuroni su zajedno grupirani u slojeve. Neuronska mreža ima jedan ulazni sloj proizvoljni broj skrivenih slojeva i jedan izlazni sloj. Struktura neuronske mreže se može vidjeti na slici Sl. 2.1 - Struktura neuronske mreže. Implementirana mreža ima ovakvu strukturu.

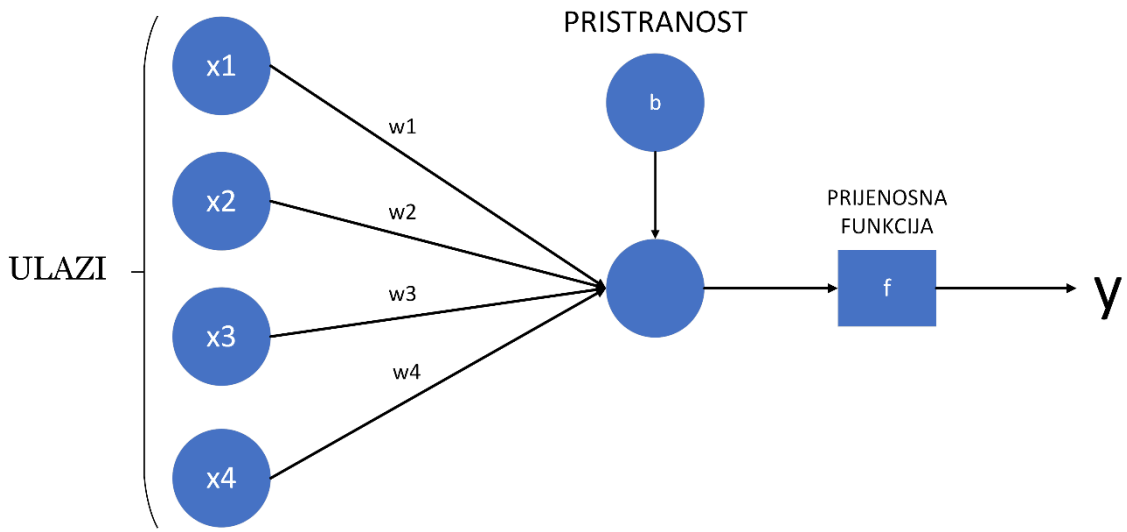


Sl. 2.1 - Struktura neuronske mreže [7]

2.1.1. Težine, pristranosti i aktivacije neuronske mreže

Svaki sloj neuronske mreže ima matricu pripadajućih težina, vektor pristranosti i vektor aktivacija. Za svaku konekciju sa prijašnjim slojem, sloj ima odgovarajuću težinu i pristranost. Sloj dobije aktivacije iz prošlog sloja kao ulaze, i treba iz njih dobiti svoje aktivacije. Prvo se matrično množi matrica težina trenutnog sloja sa vektorom aktivacija prošlog sloja. Nakon toga dobiveni vektor se matrično zbraja sa vektorom pristranosti. Na kraju je potrebno provući dobiveni vektor kroz prijenosnu funkciju kako bi dobili aktivaciju, i tada se dobivene aktivacije šalju sljedećem sloju, ako postoji. Na taj način se ulaz koji neuronska mreža dobije propagira kroz slojeve.

Povezanost sloja sa jednim neuronom se može vidjeti na slici Sl. 2.2 - Povezani sloj i neuron.



Sl. 2.2 - Povezani sloj i neuron

Ovaj proces se naravno može i prikazati jednadžbom (1).

$$g(x1, x2, x3, x4) = f(w1 \times x1 + w2 \times x2 + w3 \times x3 + w4 \times x4 + b)$$

(1)

Gdje je g(x1, x2, x3, x4) prijenosna funkcija, w1, w2, w3, w4 su težine, b je pristranost, te je f(x) prijenosna funkcija.

2.1.2. Inicijalizacija

Prilikom kreiranja nove mreže, potrebno je inicijalizirati njene vrijednosti. To se može napraviti na mnogo načina. Najjednostavniji način je uzeti uzorak brojeva iz normalne razdiobe sa nekom malom standardnom devijacijom i primijeniti ga na težine i pristranosti.

2.1.2.1 Xavier inicijalizacija

Xavier inicijalizacija[8] je korisna inicijalizacija koja uzima u obzir broj ulaza i izlaza sloja. Težine se inicijaliziraju na način da se zadrži varijanca u vrijednostima. Korisna je kada se koriste tanh[12] ili sigmoidalna[10] prijenosna funkcija sa kojima se može javiti problem sa saturacijom. Jednadžba (2) prikazuje Xavierovu inicijalizaciju.

$$W_{ij} \sim U\left[-\frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}, \frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}\right]$$

(2)

Gdje U predstavlja uniformnu razdiobu, fan_{in} broj ulaza u sloj, a fan_{out} broj izlaza iz sloja.

Kod Xavierove inicijalizacije pristranosti se postavljaju u 0.

2.1.2.2 He inicijalizacija

He ili Kaiming inicijalizacija[9] je inicijalizacija koja uzima u obzir nelinearnost prijenosnih funkcija. He inicijalizacija nastoji izbjegavati eksponencijalnu redukciju ili povećanje ulaznih signala. Često se koristi uz ReLU[15] i slične prijenosne funkcije. Jednadžba (3) prikazuje He inicijalizaciju.

$$w_l \sim \mathcal{N}\left({}^{0.2}/_{n_l}\right)$$

(3)

Gdje slovo $\mathcal{N}$ predstavlja normalnu razdiobu, a n_l broj ulaza u sloj.

Kod He inicijalizacije pristranosti se postavljaju u 0.

2.1.3. Prijenosne funkcije

Prijenosne funkcije su funkcije koje se primjene na aktivacije sloja prije nego što se te aktivacije pošalju u idući sloj. Kada se prijenosne funkcije ne bi koristile, sve operacije između slojeva bi bile linearne, te bi izlaz iz neuronske mreže bio linearna kombinacija ulaza neuronske mreže. Kako bi se izbjegla linearnost, mreže koriste nelinearne prijenosne funkcije koje primjenjuju na izlaze slojeva. Neke od najpoznatijih prijenosnih funkcija su sigmoidalna prijenosna funkcija, tanh prijenosna funkcija i ReLu prijenosna funkcija.

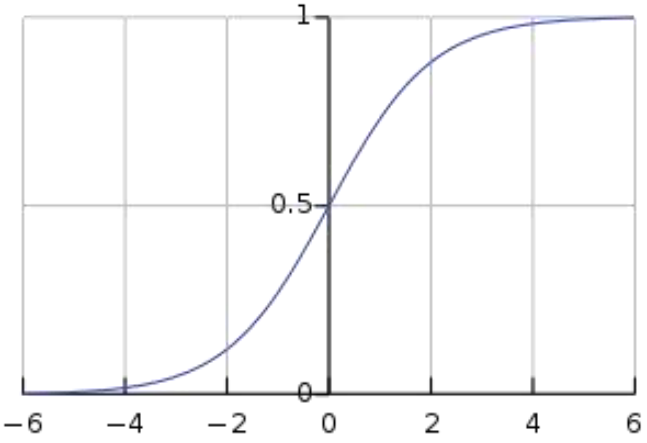
2.1.3.1 Sigmoid prijenosna funkcija

Sigmoid prijenosna funkcija je prijenosna funkcija koja koristi sigmoid funkciju[10] prikazanu u jednadžbi (4).

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{1+e^{x}} = 1 - \sigma(-x)$$

(4)

Sigmoid je funkcija koja skup realnih brojeva od $-\infty$ do $+\infty$ preslikava u skup realnih brojeva od 0 do 1. Graf ove funkcije izgleda kao na slici Sl. 2.3 - Graf sigmoid funkcije.



Sl. 2.3 - Graf sigmoid funkcije[11]

Sigmoid prijenosna funkcija je vrlo često korištena u neuronskim mrežama. Vrijednosti vrlo brzo rastu prema 1 ili 0 te se može dogoditi problem nestajućeg gradijenta [11]. Zbog niskog gradijenta na granicama učenje propagacijom unatrag postaje teže, te ako ulazi ili izlazi slojeva počnu postizati visoke ili niske vrijednosti , one se nakon prolaza u prijenosnu funkciju svode na 0 i 1 gdje gradijent nestaje. Stvara se problem saturacije te se učenje usporava.

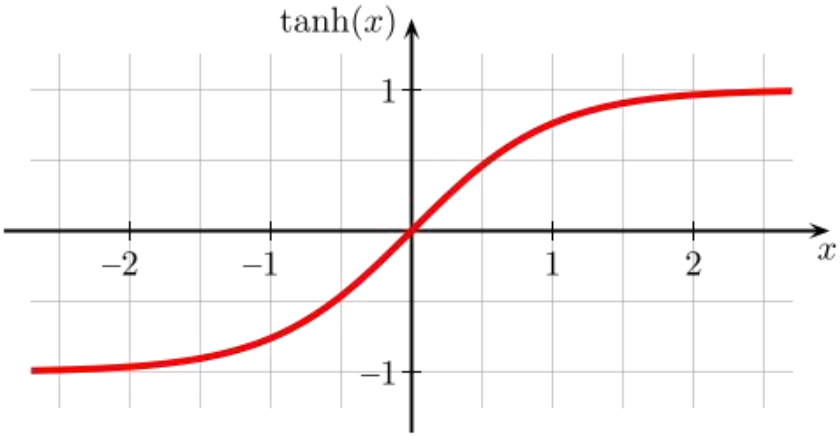
2.1.3.2 Tanh prijenosna funkcija

Tanh prijenosna funkcija koristi hiperbolnu tangens funkciju[12] koja je prikazana u jednadžbi (5).

$$\tanh(x) = \frac{sh(x)}{ch(x)} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

(5)

Hiperbolna tangens funkcija preslikava skup realnih brojeva od $-\infty$ do $+\infty$ u skup realnih brojeva od -1 do 1. Graf ove funkcije je na slici Sl. 2.4 - Graf hiperbolne tangens funkcije.



Sl. 2.4 - Graf hiperbolne tangens funkcije [13]

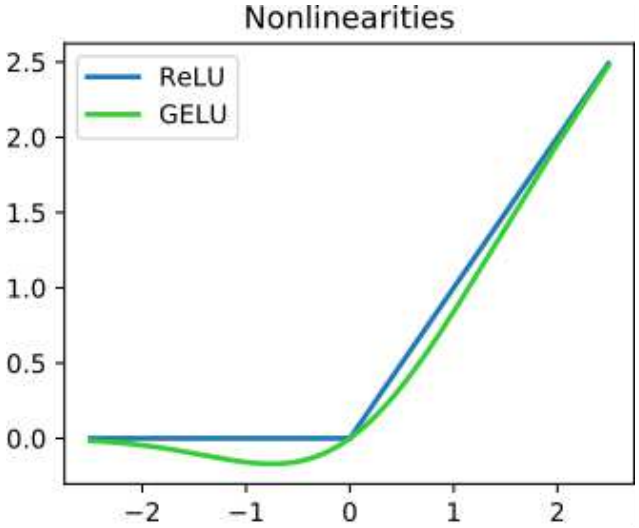
Ova prijenosna funkcija je vrlo slična sigmoid prijenosnoj funkciji. Obje imaju vrlo sličan graf, razlika je u tome što tanh preslikava realne broje u skup od -1 do 1, a sigmoid od 0 do 1. Hiperbolna tangens prijenosna funkcija pati od istih problema nestajućeg gradijenta kao i sigmoid funkcija.

2.1.3.3 ReLU prijenosna funkcija

Rectified Linear Unit (ReLU) prijenosna funkcija[14] je prijenosna funkcija koja je prvi put uvedena kao način za bolje treniranje mreža. Iako izgleda vrlo jednostavno, ova prijenosna funkcija prikazana u jednadžbi (6) je vrlo efektivna.

$$f(x) = x^{+} = \max(0, x) \tag{6}$$

ReLU prijenosna funkcija preslikava skup realnih brojeva od $-\infty$ do $+\infty$ u skup realnih brojeva od 0 do $+\infty$. Danas je ovo jedna od najčešće korištenih aktivacijskih funkcija zbog svojih prednosti. Zbog toga što je kodomena ove prijenosne funkcije od 0 do $+\infty$ problem nestajućeg gradijenta je znatno umanjen, a nelinearnost je i dalje postignuta. Graf funkcije možemo vidjeti na slici Sl. 2.5 - Graf ReLu i GELU funkcija.



Sl. 2.5 - Graf ReLu i GELU funkcija [15]

Na grafu možemo vidjeti i jednu od varijanti ReLU, *Gaussian-error linear unit*(GELU). Ova prijenosna funkcija je glatka aproksimacija od ReLU. ReLU ima još mnogo varijanti kao što su leaky ReLU, parametric ReLU, SiLU i Softplus. [15]

3. Genetski algoritam

Genetski algoritam[16] je računalni heuristički algoritam pretraživanja temeljen na idejama evolucije poput mutacije, križanja i prirodne selekcije. Koristi se za nalaženje rješenja visoke kompleksnosti za probleme optimizacije i pretraživanja. Računalnom implementacijom genetskog algoritma se mogu riješiti razni problemi uključujući i nalaženje optimalne neuronske mreže.

3.1. Karakteristike

3.1.1. Dobrota

Dobrota je karakteristika koja predstavlja kvalitetu jedinke. Što je dobrota veća to je jedinka efikasnija u rješavanju traženog problema. Dobrota u biološkom smislu predstavlja uspješnost razmnožavanja jedinke, drugim riječima jedinka koja je najkvalitetnija će ostaviti najviše kopija sebe u idućim generacijama. Kod genetskog algoritma se koristi funkcija dobrote koja vodi u smjeru optimalnog rješenja najviše dobrote. Implementacije funkcije dobrote mora biti relevantna problemu koji se pokušava riješiti.

Funkcija dobrote može biti apsolutna ili relativna. Apsolutna dobrota je brojčana vrijednost koja procjenjuje apsolutnu kvalitetu jedinke, a relativna dobrota daje relativnu procjenu, na primjer jedinka A je bolja od jedinke B.

Kako bi se izračunala dobrota često su potrebni kompleksni izračuni. Zbog računalne složenosti nekada je dovoljno izračunati procjenu dobrote, ovisno o zadatku.

3.1.2. Selekcija

Selekcija je proces odabiranja jedinke iz skupine jedinki. Pri selekciji unutar genetskog algoritma cilj je izabrati što više kvalitetnih ali različitih jedinki, kako bi održali genetsku raznolikosti u populaciji.

3.1.2.1 Roullete wheel selekcija

Roullete wheel selekcija bira jedinku sa vjerojatnošću p za tu jedinku, gdje je vjerojatnost p proporcionalna dobroti te jedinke u usporedbi sa dobrotom generacije. Formula roullete wheel selekcije je prikazana pod brojem (7).

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{7}$$

3.1.2.2 Turnir selekcija

Turnir selekcija je selekcija koja bira najbolju jedinku, iz podskupa jedinki populacije. Određujemo veličinu podskupa n, biramo n nasumičnih jedinki, te uzimamo najbolju među njima.

3.1.3. Elitizam

Elitizam je uzimanje određenog broja najboljih jedinki iz prošle generacije te direktno stavljanje tih jedinki u sljedeću generaciju.

Elitizam se koristi iz razloga što pri stvaranju nove generacije globalno optimalno rješenje može biti izgubljeno. Ako su elitističke jedinke u lokalnog optimumu može se dogoditi konvergencija u lokalni optimum i gubljenje mogućnosti nalaženja globalnog optimuma. Iz ovog razloga pri korištenju elitizma treba paziti da broj najboljih jedinki koje se prenose nije prevelik.

3.1.4. Križanje

Križanje je proces stvaranja nove jedinke iz dvije roditeljske jedinke koje se biraju selekcijskim algoritmom. Geni nove jedinke se određuju pomoću algoritma za križanje i dvije roditeljske jedinke.

Neki od algoritama križanja su uniformno križanje gdje se za svaki gen djeteta nasumično bira isti gen od jednog ili drugog roditelja, ili križanje sa točkom prijeloma gdje se bira pozicija u genomu gdje se siječe genom od oba roditelja na dva dijela te uzima po jedan dio od svakog. Također se mogu uzeti srednje vrijednosti genoma

roditelja ili uzeti gen sa vrijednošću između gena roditelja sa uniformnom ili normalnom razdiobom.

3.1.5. Mutiranje

Mutiranje je mijenjanje gena jedinki za malu vrijednost kako bi se održala genetička raznolikost. Mutiranje se obično obavlja tako da se sa zadanom vjerojatnošću mutira svaki gen, i mijenja mu se vrijednost za određeni broj koji se može uzeti ili iz normalne razdiobe sa zadanom standardnom devijacijom ili iz uniformne raspodjele sa zadanom širinom. Ako je vjerojatnost zadovoljena, tada se sa zadanom vjerojatnošću određuje hoće li mutacija biti mala ili velika. Velika mutacija je mala mutacija pomnožena sa nekim koeficijentom većim od 1.

Kod mutiranja vrijede tri generalna pravila:

- Svaka točka u prostoru traženja mora biti dohvatljiva kroz jednu ili više mutacija
- Ne smije postojati preferencija za smjer u prostoru traženja
- Malene mutacije trebaju biti vjerojatnije od velikih

Mutiranje se može dodatno unaprijediti na način da ako nije dugo došlo do promjene u dobrotama jedinki populacije, vjerojatnost mutacije se poveća i/ili iznos mutacije se poveća.

3.2. Algoritam i pseudokod

Na početku se mora inicijalizirati početna generacija kandidata. Najčešće su ti kandidati nasumično raspoređeni u domeni mogućih rješenja ili počinju oko iste točke, no mogu se uzeti i kandidati unutar dijela domene gdje znamo da postoji optimalno rješenje. Nakon inicijalizacije, svaki individualni kandidat se mora ocijeniti

dobrotom. Genetski algoritam je iteracijski algoritam. Broj iteracija je jednak maksimalnom broju generacija. Pri početku algoritma jedinke se inicijaliziraju te se stvara početna generacija. Iz te početne generacije se izračuna dobrota svake jedinke. Na temelju dobrote stvaramo novu generaciju. Prvo se elitističke jedinke prebace u novu generaciju, zatim se križaju jedinke iz prošle generacije i stvaraju nove jedinke. Kada smo ispunili novu generaciju mutiramo svaku jedinku. Ovaj proces se iterira dok maksimalni broj generacija nije dostignut ili tražena dobrota nije ostvarena. Pseudokod algoritma je prikazan u nastavku (Kod 3.1).

```
inicijaliziraj prvu generaciju
WHILE iteracija < maksimalnoIteracija  {
      izračunaj dobrotu
      novaGeneracija = stvoriNovuGeneraciju()
      mutiraj novu generaciju
      iteracija++
 }

stvoriNovuGeneraciju() {
      dodaj elitističke jedinke
      za svaku ostalu jedinku {
            izaberi dva roditelja selekcijom
            križaj roditelje i dodaj jedinku
      }
      vrati novu generaciju
```

Kod 3.1

4. Implementacija sustava

Sustav je implementiran u programskom jeziku java koristeći Maven alat za automatizaciju izrade.

4.1. Implementacija igre Texas Hold'em poker

Grafičko sučelje Texas Hold'em poker igre je implementirano u ljusci operativnog sustava. Temeljeno je na ASCII prikazu karata i tekstualnoj komunikaciji između igre i ljudskog igrača. U igri vrijede standardna pravila Texas Hold'em pokera.

4.1.1. Evaluacija karata

Računalno najzahtjevniji dio igre pokera je evaluacija snage ruke. Način na koji se evaluira ruka je pomoću hash tablica. Hash tablice su kreirane koristeći hash optimizaciju i pametno šifriranje. U toj tablici svaka kombinacija od 5 karata ima pridijeljenu cjelobrojnu vrijednost. Tablica je dodatno optimizirana na način da se kombinacije koje imaju istu snagu šifriraju na istu vrijednost. Korištenje takve tablice je vrlo efikasno i prikladno za korištenje u genetskom algoritmu koji zahtjeva brzinu. Korištena je gotova implementacija nađena na GitHub internetskoj stranici kreirana od strane korisnika Jmp. Implementacija se zove JmpEvaluator[17] te je link na GitHub stranicu implementacije priložen u literaturi.

4.2. Implementacija neuronske mreže

4.2.1. Ulazi i izlazi neuronske mreže za igru pokera

4.2.1.1 Ulazi

Poker je igra sa mnogo faktora stanja, te je mreža implementirana na način da uzima u obzir što više njih kako bi mogla donijeti apstraktnu odluku što prilagođeniju stanju za stolom.

Mreža kao ulaz dobiva sve karte na stolu i u ruci igrača neuronske mreže. Svaka karta je opisana sa 5 vrijednosti. Prva vrijednost je normalizirana vrijednost karte između 0 i

1. As ima vrijednost 1 dok dvojka ima vrijednost 2 / 14. Nakon vrijednosti, u ulazu slijede četiri broja. Svaki broj označava jednu boju. Ako karta ima boju srca, ulaz za srce je postavljen u vrijednost 1 dok su ostali ulazi 0. Karte čine 35 ulaza neuronske mreže. Kada karta na stolu nije izvučena, tada su ulazi za tu kartu svi jednaki nuli. Ulaz karte as srce je prikazan slikom u nastavku (Sl. 4.1 - Ulaz neuronske mreže za kartu as srce).

Sl. 4.1 - Ulaz neuronske mreže za kartu as srce

Nakon karte kao ulaz se šalju informacije o stanju igrača neuronske mreže. Njegov broj čipova, broj trenutnog uloga u igri, pozicija za stolom. Nakon informacija o igraču kao ulaz se šalju i informacije o stolu i drugim igračima. Broj čipova ukupno uloženih od svih igrača, pozicija small blind igrača, te za svakog igrača vrsta njegove prošle akcije i njena vrijednost u čipovima. Sa svim ovim informacijama to čini ukupno 48 ulaza. Svaki ulaz je normaliziran na vrijednosti između 0 i 1(uključivo).

4.2.1.2 Izlazi

Svaki igrač u pokeru na potezu može donijeti najviše dvije odluke. Prva odluka je potez koji želi napraviti. Taj potez ovisi o situaciji u kojoj se igrač nalazi. Mogući potezi su fold pri čemu igrač odustaje od trenutne runde, check kada je igrač uložio jednako maksimalnom ulogu te želi nastaviti igru bez dodatnih uloga, call kada igrač stavlja ulaže čipova jednako maksimalnom ulogu, raise kada igrač povisuje maksimalni ulog te allin kada igrač ulaže sve svoje čipove. Drugi potez igrač može donijeti ako je prva odluka poteza bila raise. U tom slučaju igrač mora odlučiti koliko čipova želi uložiti.

Implementirana mreža mora donijeti maksimalno dvije odluke, stoga ima dva izlaza. Prvi izlaz je klasifikacijski izlaz gdje mreža klasificira koji potez želi napraviti. Klasifikacija se može napraviti na više načina.

4.2.1.3 One hot encoding klasifikacija

Ako kategorije klasifikacije nemaju relacije jedna sa drugom tada je najbolje koristiti one hot encoding klasifikaciju. Takva klasifikacija za svaku kategoriju pridjeljuje jedan output. Gleda se koji izlaz ima najveću vrijednost i smatra se da je ulaz klasificiran u kategoriju koja pripada tom izlazu. Ova vrsta klasifikacije se koristi kada kategorije nemaju međusobne relacije, na primjer ako želimo klasificirati životinju kao vuk ili zec.

Sa druge strane klasifikacija koja je korištena za određivanje poteza je klasifikacija sa jednim izlazom. Izlaz se dijeli na kategorije, na primjer dio od 0 do 0.5 pripada kategoriji A, a dio od 0.5 do 1 pripada kategoriji B. Kategorija ulaza je jednaka dijelu u kojem se nalazi taj izlaz, na primjer ako je izlaz 0.7, kategorija je tada B. Ova vrsta klasifikacije se koristi kada između kategorija postoji neka relacija. U slučaju poteza, potez folda je bliže potezu calla nego raisea, i obrnuto.

4.2.1.4 Regresija

Kada je izabran potez koji mreža bira, ako je potez raise mora se odrediti vrijednost uloga. Uzima se drugi izlaz i ovisno o njegovoj vrijednosti ulaže se određena količina čipova. Ovaj izlaz je regresijski izlaz, što znači da određuje kontinuiranu vrijednost kao što je iznos čipova.

4.2.2. Prijenosne funkcije implementacije

U implementaciji su korištene sigmoid i ReLU prijenosna funkcija. Za skrivene aktivacije je korištena ReLU prijenosna funkcija. Za klasifikacijski izlaz je korištena sigmoid prijenosna funkcija, a za regresijski izlaz je korištena funkcija identiteta sa ograničenjem.

4.2.2.1 Problemi sa korištenjem sigmoid prijenosne funkcije

Pri ranijoj verziji implementacije sigmoid prijenosna funkcija je bila korištena kao prijenosna funkcija za skriveni sloj. Zbog problema sa funkcijom dobrote korištenjem sigmoid prijenosne funkcije došlo je do smanjenja učenja i problema saturacije. Svi izlazi su poprimali granične vrijednosti ili 0.5, i mreža je uvijek radila iste poteze. O ovom će problemu biti još diskutirano naknadno u potpoglavlju 4.4.

4.3. Implementacija genetskog algoritma

4.3.1. Implementacija algoritma računanja dobrote

Kako bi se dobrota izračunala mreže moraju igrati implementiranu igru. Sustav je implementiran sa namjerom da mreže uče igrajući jedna protiv druge. Generacija je smještena na stolove sa nasumičnim izborom sudionika za svakim stolom. Sudionici igraju partiju dok jedan igrač ne ostane sa više od nula čipova ili dok se ne odigra predodređen broj rundi. Kada su sve partije završile, mreže se opet smješta za stolove, ali sa drugačijim protivnicima, te se opet odigravaju partije. Ovaj proces se ponavlja ukupno pet puta. Namjera je da mreže imaju raznovrsnu skupinu protivnika iz cijele generacije.

4.3.2. Implementacija inicijalizacije, selekcije, križanja i mutacije

Za inicijalizaciju je korištena Xavierova inicijalizacijska metoda. Za selekciju je korištena turnirska selekcija. Za križanje je korištena metoda srednje vrijednosti.

4.3.3. Implementacija funkcije dobrote

Implementacija funkcije dobrote predstavlja najveću prepreku u treniranju neuronske mreže za poker. Poker je igra na sreću u kojoj i najgori igrač nekada pobijedi. Korišteno je više varijanti funkcija dobrote. O problemima sa svakom funkcijom dobrote biti će diskutirano u potpoglavlju 4.4.

4.3.3.1 Funkcija dobrote srednje vrijednosti količine čipova

Funkcija dobrote srednje vrijednosti dobitka čipova računa srednju vrijednost količine čipova iz svih stolova za kojima je mreža igrala. Kada igrači igraju dovoljno rundi međusobno igrač koji je najviše zaradio bi trebao biti najkvalitetniji. U pravoj igri pokera među ljudskim igračima, kada se uzme u obzir dovoljni broj partija ovo pravilo zaista procjenjuje kvalitetu igrača.

4.3.3.2 Funkcija dobrote evaluacijom poteza

Funkcija dobrote evaluacijom poteza prati svaki potez koji je određeni igrač napravio, i na kraju runde evaluira koliko je su potezi bili kvalitetni i na temelju toga daje ili oduzima igraču bodove dobrote. Na primjer, ako je igrač u jednoj rundi povisio ulog, i izgubio je rundu bodovi će mu se oduzeti. Ako je imao slabu ruku, puno će mu se više bodova oduzeti nego ako je imao snažnu ruku. Isto tako ako je igrač povisio ulog i dobio bodovi će mu se dodati, no puno više bodova će mu se dodati ako je imao snažnu ruku nego ako je imao slabu ruku. Formula za izračun bodova u slučaju kad je igrač povisio ulog i dobio je prikazana u nastavku (8).

$$f(x) = k_p * \ b_x * S_x \tag{8}$$

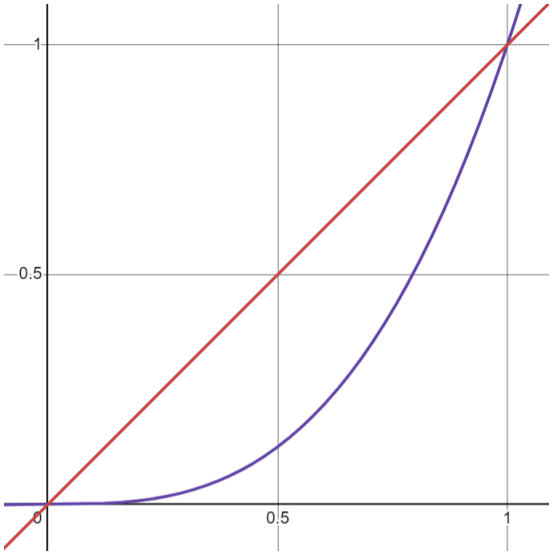
Gdje je k_p koeficijent poteza. Koeficijent poteza je koeficijent koji određuje kakve posljedice potez ima u igri. Koeficijent poteza allin-a koji ima velika posljedice je veći od koeficijenta fold-a koji nema veliki utjecaj na poziciju igrača. b_x je ulog koji je igrač uložio. S_x je snaga ruke igrača, skalirana između 0 i 1. Ovisno o potezu i rezultatu runde svaka druga jednadžba izgleda malo drukčije.

4.3.3.3 Funkcija dobrote razlike snage ruke i uloga

Funkcija razlike snage ruke i uloga računa omjer snage ruke kroz maksimalnu snagu ruke i omjer uloga kroz maksimalni ulog. Omjer snage ruke se potencira faktorom kako korelacija između snage i uloga ne bi bila linearna; za slabe do srednje ruke se ulaže malo te kada ruka postaje jača ulog naglo raste. Želi se postići da vrijedi jednadžba koja slijedi (9).

$$\left(\frac{s_x}{s_{max}}\right)^3 = \left(\frac{b_x}{b_{max}}\right) \quad (9)$$

Gdje je s_x snaga ruke igrača, s_{max} maksimalna snaga ruke, b_x ulog igrača, b_{max} maksimalni ulog igrača. Graf ove funkcije je prikazan plavom bojom na slici (Sl. 4.2 - Graf omjera snage i omjera uloga napravljen u Desmos[19] alatu), gdje je y os omjer uloga a x os omjer snaga. Crvenom bojom je prikazan linearni graf u kojem je za slabe ruke ulog previsok.



Sl. 4.2 - Graf omjera snage i omjera uloga napravljen u Desmos[19] alatu

4.4. Problemi implementacije

4.4.1. Problemi sa algoritmom računanja dobrote

Mrežama se računa dobrota dok igraju poker sa drugim mrežama. Neke mreže igraju vrlo neoptimalno i zbog toga neoptimalne strategije drugih mreža postaju pobjedničke. Zbog toga točnost podataka na kojima mreža uči postaje upitan. Ako je jedna mreža za stolom nasumično uložila sve čipove, taj ulazni podatak gubi značaj koji bi imao u pravoj partiji protiv ljudskih igrača. Mreža nema mogućnost napraviti korelacije između poteza drugih igrača, njihove mogućnosti dobitka i naposljetku svojeg odgovora. Ovaj problem se dalje širi u arhitekturu neuronske mreže gdje neki ulazi postaju šum, te u funkcije dobrote u kojima postaje potrebno procijeniti kvalitetu mreže isključivo na temelju samo njenog stanja, a ne stanja za stolom.

4.4.2. Problemi sa korištenjem sigmoid prijenosne funkcije za skrivene slojeve

Sigmoid prijenosna funkcija ima poznati problem sa saturacijom. Ova prijenosna funkcija preslikava velike pozitivne brojeve u 1, i male negativne brojeve u -1. Kada se u aktivacijama počnu pojavljivati visoki brojevi, sve se aktivacije tada preslikavaju u istu vrijednost. Tijekom propagiranja kroz sve slojeve dolazi do saturacije i sve ulazne vrijednosti se preslikavaju u iste izlaze. Zbog toga mreža nije sposobna učiti.

Kod korištenja objektivne funkcije dobrote ovaj problem se može izbjeći, jer same vrijednosti za koje neće doći do saturacije su preferirane dobrotom. No, u igrama na sreću kao što je poker teško je objektivno evaluirati dobrotu. Mreža kojoj vrijednosti težina naginju saturaciji može dobiti na sreću i tada cijela populacija naginje prema saturaciji. Kada se ova pojava ponovi više puta tijekom treniranja, cijela populacija konvergira prema vrijednostima težina i pristranosti koje izazivaju saturaciju i tada učenje prestaje.

Zbog ovog problema u implementaciji je bila potrebna promjena prijenosnih funkcija skrivenog sloja, i implementacija drugačijih funkcija dobrote koje su objektivnije.

4.4.3. Problemi sa funkcijama dobrote

4.4.3.1 Problem sa funkcijom dobrote srednje vrijednosti čipova

Prva implementirana funkcija dobrote je funkcija dobrote srednje vrijednosti čipova opisana u potpoglavlju 4.3.3.1. Problemi sa ovom funkcijom dobrote su nastali zbog utjecaja sreće u njenoj evaluaciji i načina na koji mreže međusobno igraju. Problem je direktno povezan sa problemom opisanim u potpoglavlju 4.4.1. Zbog toga što mreže igraju međusobno, dobitnička mreža je često neoptimalna, a gubitničke mreže su još neoptimalnije. Dobitak postaje loš pokazatelj kvalitete te se preciznost procjene kvalitete gubi. Mreže koje najviše ulože i posreći im se postanu „najdobrije“ te nakon mnogo iteracija sve mreže počnu što više ulagati.

4.4.3.2 Problemi sa funkcijom dobrote evaluacije poteza

Funkcija dobrote pomoću evaluacije poteza je opisana u potpoglavlju 4.3.3.2 i procjenjuje dobrotu na temelju kvalitete poteza koje mreža radi. Problem ove funkcije dobrote je u njenom kompleksnom izračunu i načinu bodovanja.

Teško je odrediti sva stanja u kojima mreža može biti i točan potez u tim stanjima. Nadalje, iako je relativno lagano odrediti kada je primjerice potez povišenja uloga bio dobar, poteze poput checka je teško vrednovati. Čak i ako uspijemo vrednovati check, teško je uskladiti sve koeficijente poteza kako bi mreža donosila dobre odluke. Ako je neki koeficijent imalo prevelik mreže će početi preferirati te poteze.

Potez povišenja uloga se vrednuje na temelju snage ruke, količine uloga i toga je li igrač pobijedio. Tu opet dolazi do poveznice sa problemom opisanim u potpoglavlju 4.4.1. Mreža koja je pobijedila nije nužno igrala dobro ali se vrednuje kao da je dobro igrala. Tada mreže koje ne rade nužno najbolje poteze postanu preferirane i konvergira se u loše rješenje.

4.4.3.3 Problemi sa funkcijom dobrote razlike omjera snage i uloga

Funkcija dobrote razlike omjera snage i uloga opisana u potpoglavlju 4.3.3.3 procjenjuje kvalitetu mreže na temelju razlike između snage ruke koju mreža ima i uloga mreže.

Ova funkcija dobrote rješava probleme koje ostale funkcije dobrote imaju. Zbog toga što se kvaliteta mreže procjenjuje samo na temelju uloga i jačine ruke koji je mreža ostvarila u rundi, a ne uzima se u obzir pobjeda i osvojeni čipovi te generalno nema usporedbe sa ostalim mrežama, postiže se objektivnije mjerenje kvalitete. Iako ova funkcija dobrote ne uzima u obzir kompleksnije odnose stanja igre i poteza, lakša je za koristiti i preciznija je u evaluaciji.

Zbog toga što ne uzima u obzir kompleksnije odnose stanja igre, pri korištenju ove funkcije dobrote većina ulaza u neuronsku mrežu postaje redundantno. Mreža se procjenjuje samo na temelju snage ruke i uloga, stoga jedini parametri koje mreža treba primiti na ulaz su karte u ruci i na stolu, iznos trenutnog uloga te ukupni iznos čipova koje mreža ima. Svi ostali ulazi postaju šumovi koji mreži usporavaju učenje.

5. Moguće nadogradnje

5.1. Implementacija neuronske mreže jednostavne arhitekture

Uz funkciju dobrote razlikom omjera snage ruke i uloga (4.3.3.3) implementirana mreža je prekompleksna što otežava učenje. Implementiranje mreže jednostavnije arhitekture sa manje ulaza bi možda ubrzalo učenje i genetskim algoritmom bi se možda našlo bolje rješenje.

Ovakva jednostavna arhitektura bi primala samo karte igrača, karte na stolu, iznos uloga i ukupni iznos čipova igrača u obzir. Takva mreža bi imala manje šumova te je manja vjerojatnost konvergencije prema neoptimalnom rješenju.

5.2. Implementacija treniranja mreže jačinom karata u ruci

Trenutno implementirane mreže ulaze u igru protiv drugih mreža sa nula znanja o igri. Potencijalno poboljšanje je treniranje mreža na podatcima o početnim kartama u ruci i idealnim potezima na temelju postojećih tablica.

Ovakve mreže bi ušle u igru sa izgrađenim početnim znanjem. Implementacija predtreniranja jačinom karata u ruci bi potencijalno riješila probleme sa algoritmom dobrote i funkcijama dobrote koji su opisani u poglavlju 4.4.

5.3. Implementacija unaprijeđenih funkcija dobrote

Implementirane funkcije dobrote zbog opisanih problema ili procjenjuju loše ili ne uzimaju cijelo stanje igre u obzir. Potencijalno unaprijeđenje sustava bi bila nova implementacija funkcije dobrote koja dobro procjenjuje i uzima cijelo stanje igre u obzir. Šumovi bi nestali i učenje bi se ubrzalo.

Potencijalni problem ovakve funkcije dobrote predstavlja kompleksnost izračuna.

Zaključak

Cilj rada je bio implementirati igru Texas Hold'em poker te neuronske mreže i genetski algoritam u kojem mreže uče igrati na način da igraju međusobno. Zbog nedostataka u računanju dobrote i inherentnoj sreći koja je dio igre pokera, pametni igrač koji je dobiven kao rezultat nije efikasan. Čini mnogo pogrešnih poteza i igra između ovakvih agenata se svodi na sreću.

Problemi implementacije su objektivno i precizno određivanje dobrote te način učenja međusobnom igrom u kojemu može doći do pogrešne interpretacije stanja.

Kako bi napravili kvalitetnog agenta, potrebna su poboljšanja. Neka od potencijalnih poboljšanja su treniranje neuronskih mreža na skupu podataka o igri, na primjer skupu podataka o kartama u rukama i optimalnim prvim potezima, te tek nakon toga treniranje međusobnom igrom. Implementacija jednostavnije arhitekture neuronske mreže koja kao ulaz dobiva osnovne informacije o stanju igrača kojim mreža upravlja ima potencijala biti efektivna, no ne može reagirati na poteze drugih igrača i slične apstrakcije u stanju igre.

Literatura

[1] Wikipedia, *Positions(poker)*, Poveznica: https://en.wikipedia.org/wiki/Position_(poker)

[2] Wikipedia, *Dobitne kombinacije u pokeru,* Poveznica: https://bs.wikipedia.org/wiki/Dobitne_kombinacije_u_pokeru

[3] Sklansky, D., *The Theory of Poker* (1999.)

[4] Sveučilište Cornell blog, *Game Theory Optimal(GTO) Texas Holdem Poker Theory.* Poveznica: https://blogs.cornell.edu/info2040/2021/11/03/game-theory-optimal-gto-texas-holdem-poker-theory/

[5] Elridge, S., *Nash Equillibrium,* Encyclopedia Brittanica, Poveznica: https://www.britannica.com/science/Nash-equilibrium

[6] Wikipedia, *Pluribus(poker bot),* Poveznica: https://en.wikipedia.org/wiki/Pluribus_(poker_bot)

[7] Wikipedia, *Umjetna neuronska mreža,* Poveznica: https://hr.wikipedia.org/wiki/Umjetna_neuronska_mre%C5%BEa

[8] Papers with code, *Xavier initialization,* Poveznica: https://paperswithcode.com/method/xavier-initialization

[9] Papers with code, *Kaiming initialization,* Poveznica: https://paperswithcode.com/method/he-initialization

[10] Wikipedia, S*igmoid function,* Poveznica: https://en.wikipedia.org/wiki/Sigmoid_function

[11] Jacob, T., KDnuggets, *Vanishing Gradient Problem: Causes, Consquences, and Solutions,* Poveznica: https://www.kdnuggets.com/2022/02/vanishing-gradient-problem.html

[12] Wikipedia, *Hiperbolne funkcije,* Poveznica: https://hr.wikipedia.org/wiki/Hiperbolne_funkcije

[13] Wikimedia commons, *Graf hiperbolne tangens funkcije,* Poveznica: https://commons.wikimedia.org/wiki/File:Hyperbolic_Tangent.svg

[14] Krishnamurthy, B., *An Introduction to the ReLu Activation Function,* Builtin, Poveznica: https://builtin.com/machine-learning/relu-activation-function

[15] Wikipedia, *Rectifier(neural networks),* Poveznica: https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

[16] ScienceDirect, *Genetic algorithm,* Poveznica: https://www.sciencedirect.com/topics/engineering/genetic-algorithm

[17] Jmp, *JmpEvaluator*, *(2020.),* Github, Poveznica: https://github.com/jmp/poker-hand-evaluator/tree/master

[18] EITCA, *What is one hot encoding?,* (2024.), Poveznica: https://eitca.org/artificial-intelligence/eitc-ai-dltf-deep-learning-with-tensorflow/tensorflow-deep-learning-library/tflearn/what-is-one-hot-encoding-2/

[19] Desmos, *Graphing calculator,* Poveznica: https://www.desmos.com/calculator

Privitak

Upute za korištenje programske podrške

Programska podrška se nalazi na poveznici: https://github.com/Frans1905/PokerAI.

Za korištenje softvera potreban je alat Maven. Upute za instalaciju Maven alata: https://maven.apache.org/install.html.

Kako bi pokrenuli sofver prvo je potrebno klonirati repozitorij sa GitHub-a. Pozicionirajte se u željeni direktorij i upišite u terminal:

```
git clone https://github.com/Frans1905/PokerAI.git
```

Nakon kloniranja repozitorija, pozicionirajte se u root direktorij projekta i unesite sljedeću naredbu u terminal:

```
mvn install compile
```

Programska podrška je sada spremna za korištenje.

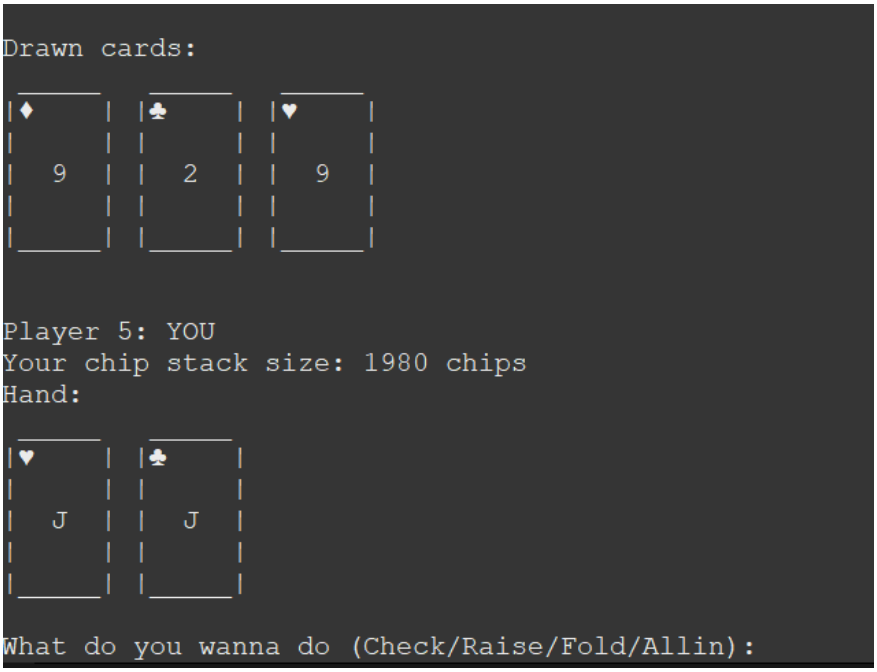
Za pokretanje aplikacije u root direktoriju unesite komandu:

```
mvn exec:java "-Dexec.mainClass=game.app.PokerAI"
```

Pri pokretanju susrest ćete se sa korisničkim sučeljem na slici Sl. 0.1 - Korisničko sučelje aplikacije.

Sl. 0.1 - Korisničko sučelje aplikacije

Za odabir opcije mora se unijeti odgovarajući broj. Prva opcija demo započinje igru protiv tri demo igrača. Jedan igrač uvijek povisuje ulog, drugi igrač uvijek folda, i treći igrač uvijek zove ulog. Druga opcija, train network, započinje treniranje mreže genetskim algoritmom. Svaka generacija je ispisana zajedno sa vremenom koje je prošlo te srednjom, medijalnom i najvećom dobrotom. Na kraju treniranja je moguće spremiti mrežu u datoteku te igrati protiv mreže. Treća opcija započinje igru protiv postojeće mreže. Odabire se broj igrača, te se za svako slobodno mjesto za stolom odabire jedna od spremljenih mreža. Nakon odabira svih mreža igra prikazana na slici(Sl. 0.2 - Korisničko sučelje igre) počinje.



Sl. 0.2 - Korisničko sučelje igre

U korisničkom sučelju na vrhu su ispisane karte na stolu. Slijedi redoslijed akcija prije igračevog poteza. Nakon akcija ispisano je igračevo stanje; njegov broj čipova i karte u ruci. Na kraju se igrača traži tekstualni unos poteza.

Sažetak

Implementirana je neuronska mreža trenirana genetskim algoritmom za igranje igre Texas Hold'em poker. Implementirani način treniranja je međusobnom igrom. Način treniranja i računanja dobrote kroz takvu igru predstavljaju problem zbog karakteristike Pokera kao igre na sreću. Dobiveni agent ne radi najoptimalnije poteze. Potencijalna poboljšanja su treniranje podatcima prije igre, promjena arhitekture neuronske mreže ili promjerna računanja dobrote.

Neural network which was trained using a genetic algorithm to play Texas Hold'em poker was implemented. The implemented way of training was playing against each other. Caculating fitness combined with the unoptimal way of training was a problem because of characteristics of poker as a game of luck. The obtained agent did not make the most optimal moves. Potential improvements include training on existing data before playing with each other, changing network architecture and changing the way fitness is calculated.